OPP NOTE

Java Thread built in methods

জাভাতে কিছু নিজস্ব *method* আছে যে গুলো *Thread* এর *implementation* এর জন্য ব্যবহার করা যায়ঃ

- ***getName*** *Thread* এর নাম *return* করে

getName()
➤ Returns the name of the thread.

- ***getPriority*** *Thread*-এর *priority return* করে

getPriority()
➤ Returns the priority of the thread.

- ***isAlive*** *Thread* চলমান আছে কিনা যাচাই করে\

isAlive()
➤ Checks if the thread is still running.

- ***join*** *Thread* এর শেষ হওইয়া পর্যন্ত অপেক্ষা করে

join()
➤ Waits for the thread to finish before continuing.

- ***run*** *Thread* শুরু হওয়ার মেথড

run()
➤ This method contains the code to run when the thread starts.

- ***sleep*** *Thread* একটি নির্দিষ্ট সময় পর্যন্ত বন্ধ থাকে

sleep(time)
➤ Pauses the thread for a specific time.

- ***start*** *Thread* শুরু হয় এই মেথড কল করলে

start()

➤ Starts the thread and calls the `run()` method.

Java Thread তৈরির উপায়

Thread মুলত দুইভাবে তৈরী করা যায়ঃ

- Runnable Interface কে implement করে
- Thread class কে extend করে

Ways to Create a Java Thread

A thread in Java can mainly be created in two ways:

1. By implementing the **Runnable** interface
2. By extending the **Thread** class

Runnable Interface

১। প্রথমে একটা ক্লাস বানাতে হবে যেটা Runnable Interface এর run() মেথড কে implement করবে।

```
class RunnableImpl implements Runnable {\
```

```
public void run(){
```

```
            System.out.println("run");     }


}
```

২। এরপর Thread এর একটা object তৈরী করতে হবে এবং RunnableImpl class কে constructor এ pass করতে হবে।

```
Thread thread = new Thread(new RunnableImpl());
```

৩। তারপর আমাদেরকে thread object টা চালাতে হবে start() মেথড কল করে।

```
thread.start();
```

Thread Class

১। প্রথমে একটা ক্লাস বানাতে হবে যেটা Thread class কে extend করবে।

```
class ThreadImpl extends Thread {


     public void run(){


System.out.println("run");
```

```
        }


}
```

২। এরপর Thread এর একটা object তৈরী করতে হবে।

```
ThreadImpl thread = new ThreadImpl();
```

৩। তারপর আমাদেরকে thread object টা চালাতে হবে start() মেথড কল করে।

```
thread.start();
```

জাভা কনকারেন্সিঃ মাল্টিথ্রেডিং

যখন একাধিক থ্রেড পাশাপাশি চলতে থাকে তখন তাকে মাল্টিথ্রেডিং বলে।

Extending Thread Class

এখানে থ্রেড ক্লাস এক্সটেন্ড করে মাল্টিথ্রেডিং ডিজাইন করা হয়েছে।

```
class Runner extends Thread {
```

```
    @Override

    public void run() {

        for (int i = 0; i < 5; i++) {

            System.out.println("Hello: " + i + "
Thread: " + Thread.currentThread().getName());

            try {

                Thread.sleep(100);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

    }

}
```

```
public class ApplicationExtends {

   public static void main(String[] args) {

        Runner runner1 = new Runner();

        runner1.start();

        Runner runner2 = new Runner();

        runner2.start();

   }

}
```

Implementing Runnable Interface

এখানে runnable ইন্টারফেস ইমপ্লিমেন্ট করে মাল্টিথ্রেডিং ডিজাইন করা হয়েছে।

```
class Runner implements Runnable {
```

```
    @Override

    public void run() {

        for(int i=0; i<5; i++) {

            System.out.println("Hello: " + i + "
Thread: " + Thread.currentThread().getName());

            try {

                Thread.sleep(100);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

    }

}
```

```
public class ApplicationRunnable {

   public static void main(String[] args) {

        Thread thread1 = new Thread(new Runner());

        Thread thread2 = new Thread(new Runner());

        thread1.start();

        thread2.start();

   }

}
```

Using Lambda Expression

এখানে ল্যামডা ফাংশন ব্যবহার করে মাল্টিথ্রেডিং ডিজাইন করা হয়েছে।

```
public class Application {
```

```
   public static void main(String[] args) {

        Thread thread = new Thread(new Runnable() {

            @Override

            public void run() {

                for (int i = 0; i < 5; i++) {

                    System.out.println("Hello: " + i +
" Thread: " + Thread.currentThread().getName());

                    try {

                        Thread.sleep(100);

                    } catch (InterruptedException e) {

                        e.printStackTrace();
```

```
                    }

                }

            }

        });

        Runnable runnerLambda = () -> {

            for (int i = 0; i < 5; i++) {

                System.out.println("Hello: " + i + "
Thread: " + Thread.currentThread().getName());

                try {

                    Thread.sleep(100);

                } catch (InterruptedException e) {
```

```
                    e.printStackTrace();

                }

            }

        };

        thread.start();

        new Thread(runnerLambda).start();

    }

}
```

জাভা কনকারেন্সিঃ ভোলাটাইল

Volatile Keyword

volatile কি ওয়ার্ড ব্যবহার করা হয় অবজেক্ট আর প্রিমিটিভ টাইপের জন্য। এটা ব্যবহার করা হয় বিভিন্ন থ্রেডে variable এর মান আপডেট করার ক্ষেত্রে। ক্লাসকে থ্রেড সেফ করার জন্য ও এটি ব্যবহার করা হয়। থ্রেড সেফ বলতে বোঝায় একাধিক থ্রেড একটা ক্লাসের মেথড এবং প্যারামিটার একই সময়ে এক্সেস করতে পারে। volatile কি ওয়ার্ডের ভ্যালু লোকাল cache এ সেভ করে না, এটা সেভ হয় সরাসরি main memory তে। যদি আমরা এমন একটা প্রোগ্রাম লিখি যেখানে দুটি থ্রেড আছে একটি থ্রেড নির্দিষ্ট ভ্যারিয়েবলের ভ্যালু read করে আর অন্যটি write করে। এখানে যদি ভ্যারিয়েবল volatile না হয় তাহলে একটি থ্রেডে write হলে অন্যটিতে read করতে পারে না।

```
public class ReaderWriterVolatile {

   private static volatile int counter;

   public static void main(String[] args) throws InterruptedException

   {

       Thread thread1 = new Thread(() -> {

           int temp = 0;

           while (true) {

               if (temp != counter) {

                   temp = counter;
```

```
                    System.out.println("reader: value of c = " + counter);

                }

            }

        });



        Thread thread2 = new Thread(() -> {

            for (int i = 0; i < 5; i++) {

                counter++;

                System.out.println("writer: changed value to = " + counter);

                try {

                    Thread.sleep(1000);

                } catch (InterruptedException e) {

                    e.printStackTrace();
```

```
                }

            }

            // sleep enough time to allow reader thread to read pending changes
(if it can!).

            try {

                Thread.sleep(5000);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

            //exit the program otherwise other threads will be keep waiting on c
to change.

            System.exit(0);

        });
```

```
        thread1.start();

        thread2.start();

    }

}
```

Output with volatile keyword

```
writer: changed value to = 1

reader: value of c = 1

writer: changed value to = 2

reader: value of c = 2

writer: changed value to = 3

reader: value of c = 3

writer: changed value to = 4
```

```
reader: value of c = 4
```

```
reader: value of c = 5
```

```
writer: changed value to = 5
```

Output without volatile keyword

```
writer: changed value to = 1
```

```
reader: value of c = 1
```

```
writer: changed value to = 2
```

```
writer: changed value to = 3
```

```
writer: changed value to = 4
```

```
writer: changed value to = 5
```

জাভা কনকারেন্সিঃ সিঙ্ক্রোনাইজ

মাল্টিথ্রেডিং এর ক্ষেত্রে একাধিক থ্রেড একটি নির্দিষ্ট মেথড একই সময়ে কল করে তখন তাকে রেস কন্ডিশন(Race Condition) বলে। এইটাকে avoid করার জন্য সিঙ্ক্রোনাইজেশন করা হয়।
সিঙ্ক্রোনাইজেশন করা হলে একটি মেথডে এক সময়ে একটি থ্রেড প্রবেশ করতে পারবে এবং কাজ শেষ হলে অন্যটা প্রবেশ করবে। সিঙ্ক্রোনাইজেশন করতে হলে কোন মেথডের আগে synchronized কি ওয়ার্ড ব্যবহার করতে হয়।

```
public class Synchronization {

   private static int count = 0;

   public static synchronized void increment(){

       count++;

   }

   public static void main(String[] args) {

       Thread thread1 = new Thread(new Runnable() {

           public void run() {

               for (int i = 0; i < 10000; i++) {
```

```
                    increment();

                }

            }

        });

        thread1.start();



        Thread thread2 = new Thread(new Runnable() {

            public void run() {

                for (int i = 0; i < 10000; i++) {

                    increment();

                }

            }

        });
```

```
        thread2.start();



        try {

            thread1.join();

            thread2.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }



        System.out.println("Count is: " + count);

    }

}
```